第2章
FPGAの専用機能ブロックを生かした回路方式
――Design Wave設計コンテスト 2006 Professional部門 第1位

田中宏昌

Design Wave設計コンテスト2006のProfessional部門第1位の設計を紹介する．仕様書で示された繰り返し型のアルゴリズムは，繰り返し回数が多ければエラー・レートが良くなるとは限らない．そこでC言語によりアルゴリズムの検討を行い，アーキテクチャを決定した．回路設計では面積優先の方針をとり，FPGAの専用機能ブロック（クロック管理，メモリ）を活用した．メモリ・ブロックを2プレーンに分割し，データ受信と復号処理を並列処理させる．また，性能向上のため，DCM（Digital Clock Manager）により外部クロックを8てい倍したクロックをFPGA内部で使用する．なお，本稿で紹介する回路の設計データは本誌のWebサイト（http://www.cqpub.co.jp/dwm/）からダウンロードできる．　（編集部）

「今年の課題はエラー訂正か…」．

昨年のコンテスト（Design Wave設計コンテスト2005）で第2位になり，賞品としてmac miniをもらって味をしめていた私は，今年も参加しようと決めていました．しかしエラー訂正はハードウェアでもソフトウェアでも経験がなく，大学の授業で1～2コマ学んだ程度です．仕様書(1)を読んでも，何をやっているのかピンときません．

そこでまずはとっかかりとして，仕様書で紹介されている繰り返し方式のアルゴリズムをC言語で書いてみました．ソース・コードは200行程度でした．動作させてみると，$S/N=6$dBと$S/N=9$dBのサンプル・データではエラーが完全に修正されます．

「なるほど，これがエラー訂正か」

少し感動です．

1．アルゴリズムの検討

C言語プログラムの動作には，一つ気になることがありました．繰り返し回数を増やしても，エラー数が単調に減少しないのです．繰り返し回数を増やすと，あるところまではエラーが減少するのですが，それを超えると逆にエラーが増加しています．エラー・レートが単調減少して一定値に収束するのであれば，その収束値を復号性能の限界の目安にできたのですが…．そこで理論を知る必要性を感じ，設計仕様書の数式をトレースしてみることにしました．

● 基準アルゴリズムを作る

コンテストの仕様では，ガウス・ノイズが仮定されています．受信信号は，ノイズによって値が分散します．受信側では，受信信号から事後値を求め，その値から送信信号を推定します．

仕様書には，送信データのビットごとに事後値を求める手順が示されています．ビットごとに計算しているので，想定される送信データは‘0’か‘1’の2通りです．事後値も2個ですが，大小関係を比較するには一方がわかれば十分です．

別の視点で，4ビットの送信データをひとかたまりとみなすこともできます．その場合，想定される送信データの組み合わせは16通りになります．事後値も16個になるので，それを求めるための計算量は増えます．

復号アルゴリズムは次のようになります．なお，厳密には

KeyWord　エラー訂正，C言語，FPGA，2ポートRAM，Spartan-3，Write after Read，DCM

事後値ではなく，事後値の未知の正定数倍を計算しています．

送信データ：(b_3, b_2, b_1, b_0)
送信信号　：$\{S_{y0}, S_{y1}, S_{y2}, S_{y3}, S_{r0}, S_{r1}, S_{c0}, S_{c1}\}$
受信信号　：$\{R_{y0}, R_{y1}, R_{y2}, R_{y3}, R_{r0}, R_{r1}, R_{c0}, R_{c1}\}$

1）送信データの候補は，4ビットなので16通りある．すべてについて事後値$\{y_{p(0000)}, y_{p(0001)}, y_{p(0010)}, y_{p(0011)}, \cdots, y_{p(1111)}\}$を求める．
2）16個の事後値の中から受信信号ともっとも近い（似ている）ものを探索し，送信データを推定する．ノイズがない理想的な伝送路の場合，該当するデータの事後値は0になる．

以降，これを基準アルゴリズムと呼ぶことにします．繰り返し処理を含まないため，収束性によるあいまいさが発生せず，事後値が一意に定まります．

基準アルゴリズムについてエラー・レートがいくつになるかC言語でシミュレーションしました．受信信号には配布されたデータ20,000個（パリティを含む）を使用しました．結果を**表1**の右列に示します．このエラー・レートをもとにほかのアルゴリズムを評価します．

● 繰り返しアルゴリズムを再評価

あらためて仕様書で紹介された繰り返し方式のアルゴリズムについて性能を評価してみました．アルゴリズムは次のようになります．ビットごとの外部値を$\{y_{0e}, y_{1e}, y_{2e}, y_{3e}\}$，事後値を$\{y_{0p}, y_{1p}, y_{2p}, y_{3p}\}$とし，受信信号は基準アルゴリズムに準じます．

1）外部値$y_{0e}, y_{1e}, y_{2e}, y_{3e}$に初期値を与える．

$$y_{0e}(0) = \mathrm{ext}(R_{y1}, R_{r0})$$
$$y_{1e}(0) = \mathrm{ext}(R_{y0}, R_{r0})$$
$$y_{2e}(0) = \mathrm{ext}(R_{y3}, R_{r1})$$
$$y_{3e}(0) = \mathrm{ext}(R_{y2}, R_{r1})$$

2）次式を必要な回数計算して，外部値$\{y_{0e}(n), y_{1e}(n), y_{2e}(n), y_{3e}(n)\}$を求める．

$$y_{0e}(2k+1) = \mathrm{ext}(\{R_{y2}+y_{2e}(2k)\}, R_{c0})$$
$$y_{1e}(2k+1) = \mathrm{ext}(\{R_{y3}+y_{3e}(2k)\}, R_{c1})$$
$$y_{2e}(2k+1) = \mathrm{ext}(\{R_{y0}+y_{0e}(2k)\}, R_{c0})$$
$$y_{3e}(2k+1) = \mathrm{ext}(\{R_{y1}+y_{1e}(2k)\}, R_{c1})$$

$$y_{0e}(2k) = \mathrm{ext}(\{R_{y1}+y_{1e}(2k-1)\}, R_{r0})$$
$$y_{1e}(2k) = \mathrm{ext}(\{R_{y0}+y_{0e}(2k-1)\}, R_{r0})$$
$$y_{2e}(2k) = \mathrm{ext}(\{R_{y3}+y_{3e}(2k-1)\}, R_{r1})$$
$$y_{3e}(2k) = \mathrm{ext}(\{R_{y2}+y_{2e}(2k-1)\}, R_{r1})$$

3）事後値$\{y_{0p}(n), y_{1p}(n), y_{2p}(n), y_{3p}(n)\}$を求める．

$$y_{0p}(n) = R_{y0}+y_{0e}(n-1)+y_{0e}(n)$$
$$y_{1p}(n) = R_{y1}+y_{1e}(n-1)+y_{1e}(n)$$
$$y_{2p}(n) = R_{y2}+y_{2e}(n-1)+y_{2e}(n)$$
$$y_{3p}(n) = R_{y3}+y_{3e}(n-1)+y_{3e}(n)$$

4）事後値の符号からビットを判定する．

繰り返し方式のアルゴリズムには，いくつかのバリエーションが考えられます．以下の項目を変えて，先ほどと同

表1　C言語プログラムによって得られたエラー・レート

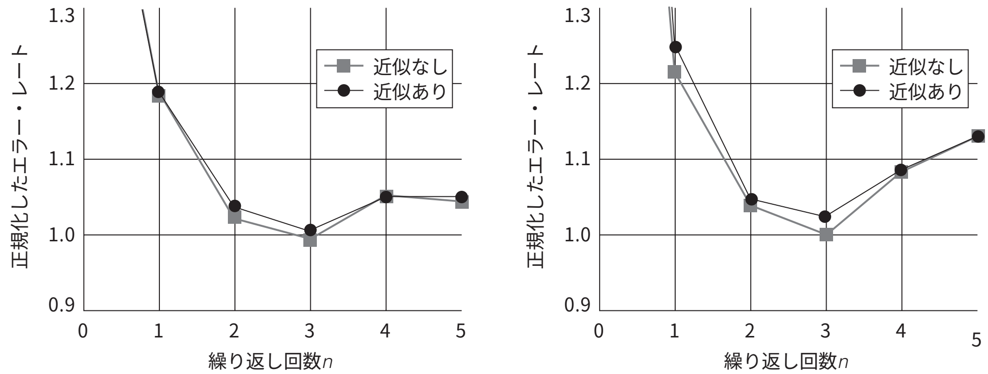
繰り返し回数n	0	1	2	3	4	5	基準アルゴリズム
S/N＝0db	0.1167	0.0832	0.0719	0.0697	0.0737	0.0731	0.0701
S/N＝3db	0.0347	0.0152	0.0130	0.0125	0.0136	0.0143	0.0125
S/N＝6db	0.0043	0.0004	0.0002	0.0002	0.0002	0.0002	0.0002
S/N＝9db	0.0002	0	0	0	0	0	0

（**a**）反復方式のアルゴリズム，ext関数近似なし

繰り返し回数n	0	1	2	3	4	5	基準アルゴリズム
S/N＝0db	0.1168	0.0833	0.0728	0.0704	0.0735	0.0735	0.0701
S/N＝3db	0.0345	0.0156	0.0131	0.0127	0.0136	0.0143	0.0125
S/N＝6db	0.0045	0.0004	0.0002	0.0002	0.0002	0.0002	0.0002
S/N＝9db	0.0002	0	0	0	0	0	0

採用

（**b**）反復方式のアルゴリズム，ext関数近似あり



(**a**) S/N = 0dB　　(**b**) S/N = 3dB

図1
繰り返し回数と正規化したエラー・レート
繰り返し回数 n = 3 のとき，エラー・レートが最小．ext 関数の近似のありなしはあまり影響がない．

じようにシミュレーションしました．

● ext関数の近似の有無

近似なし： $\mathrm{ext}(a, b) = \ln[\{1 + \exp(a \cdot b)\}/\{\exp(a) + \exp(b)\}]$

近似あり： $\mathrm{ext}(a, b) = \mathrm{sign}(ab) \cdot \min(|a|, |b|)$

● 繰り返し回数 n

結果を**表1**に示します．繰り返し回数と正規化したエラー・レートをプロットしたものを**図1**に示します．ここで正規化したエラー・レートとは，エラー・レートを基準アルゴリズムのエラー・レートで除した値です．基準アルゴリズムと同性能なら1になります．

結果を見ると，繰り返し回数 $n = 3$ のときにエラー・レートが最小となっています．また，ext関数の近似の有無はエラー・レートに1%程度しか影響していません．よって計算が簡単なext関数近似あり，繰り返し回数 $n = 3$ を採用することにしました．採用した方式はエラー・レートが基準アルゴリズムと比べて0dBで0.4%，3dBで1.6%悪化しています．ただしこの評価結果はサンプル・データから求めた実験値であって，理論値ではないことに留意します．

基準アルゴリズムと繰り返し方式以外のアルゴリズムについても良いものがないかと考えてみました．しかし，確率論による裏づけがあって，かつ繰り返し方式より演算量が少ないアルゴリズムを1ヵ月で考案するのは困難だと思い，深追いしませんでした．

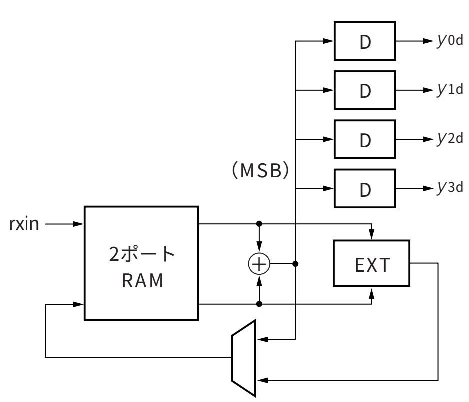


図2　採用したアーキテクチャ
二つの演算（加算とext）の組み合わせで実現している．演算器にデータを出し入れするメモリを付けて，復号処理を行う．

2. 設計方針の決定

次に回路の設計目標をどうするか考えました．面積（回路規模）を重視する場合，常とう手段は演算器の再利用です．面積はゼロになることはないので，理論的にわかるかどうかは別として，下限値が存在します．速度を重視する場合，常とう手段は演算器を複数使って並列化・パイプライン化することです．極端な話，繰り返し式をすべて展開し，すべてに回路を割り当てることもできます．速度についても，デバイス（製造プロセス）の性質などによって限界があります．

仕様では速度と面積について制約が与えられておらず，設計者に任されています．実際の設計では両者のバランスを考えるべきです．しかし，何の指標もない中で目標とするバランスを決めることはできないので，今回は面積の最小化に目標を絞りました．

● 面積最小化を目標にアーキテクチャを設計

まず，必要最小限の演算器が何であるかを考えました．アルゴリズムの中に現れる演算は，次の形をしています．

外部値　$Y = \mathrm{ext}((A + B),\ C)$

事後値　$Y = A + B + C$

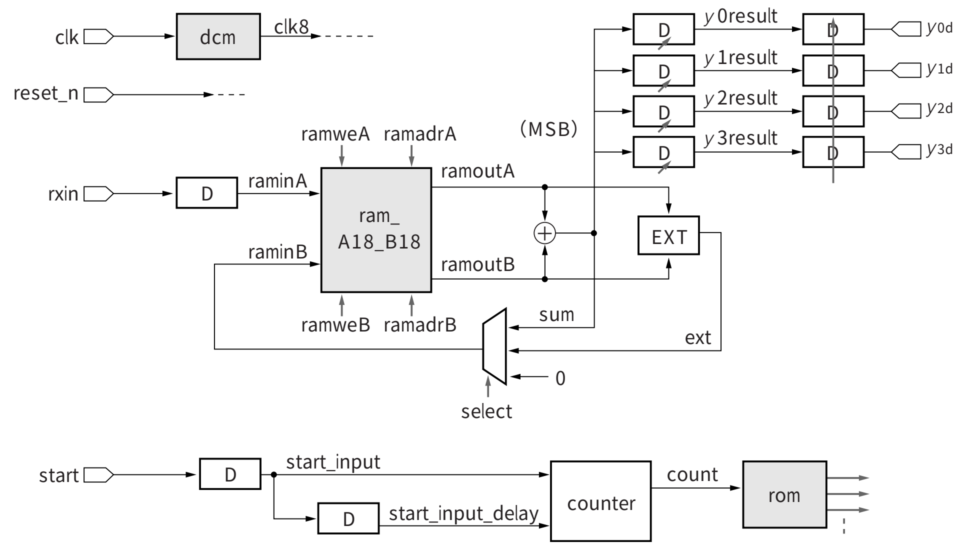


図3
復号回路のブロック図
灰色のブロックには，Spartan-3の専用機能ブロックを使用する．灰色の矢印は，カウント値から演算系の制御信号．

これらの演算は2種類の2項演算$A+B$，ext(A，B)の組み合わせで実現できます．ext関数はさらに絶対値や最小値などの関数に分解できますが，分解しても回路の共用化につながりそうにないのでこの単位でまとめています．これらの演算器にデータを出し入れするメモリ(RAM)を付けてやれば，一連の復号の計算ができそうです．メモリには2ポートRAMを使うとデータをスムーズにやり取りできます．こうして**図2**のようなアーキテクチャが完成しました．

おもなデータの流れは次のようになります．シリアルの受信データは，2ポートRAMの一方の端子から書き込みます．2ポートRAMから2個のデータを出力し，加算とext演算を行います．演算結果の一方をセレクタで選択し，2ポートRAMに格納します．この繰り返しで復号演算を行います．演算の最後に加算器から事後値を出力し，MSBをレジスタに格納して復号結果を得ます．

図2ではクロックが明示されていません．最小限の演算器を使い回そうというアーキテクチャなので，その分高速なクロックで動作させる必要があります．速度の面ではあまり高い性能は望めません．

● ターゲット・デバイスの選定

ターゲット・デバイスとして，米国Xilinx社のSpartan-3を採用しました．理由は，内蔵メモリ・ブロックでWrite after Read動作する2ポートRAM[(2)]を実現可能だったからです．Write after Read動作は，あるアドレスに書き込みを行うとき，書き込む以前のそのアドレスのデータを出力ポートに出力する動作のことです．この動作を利用することにより，演算に必要なデータの読み出しと，演算結果の格納を同時に行うことができます．逆にWrite after Read動作に対応していない場合，演算1回にデータの読み出しサイクルと書き込みサイクルが必要となり，2サイクルを要します．

3．回路の設計

アーキテクチャとターゲット・デバイスが決定したので，いよいよ回路設計です．アーキテクチャのところでは考慮しなかったRAMの制御信号(アドレス，R/W)やタイミング関係などを詰めていきました．

設計した復号回路のブロック図を**図3**に示します．図中の灰色のブロックは，Spartan-3の専用機能ブロックです．この復号回路は役割によって，クロック系，同期系，制御系，演算系に分けることができます．

クロック系では，DCM[(3)]で外部クロックを8てい倍し，内部クロックclk8を生成します．DCMの設計にはISE 7.1の付属ツール Architecture Wizardを使用しました．

同期系では，シリアルのstart信号を遅延させてエッジ検出し，カウンタを同期させます．エッジ検出が必要なのはカウンタが8てい倍されたクロックで動作するためです．エッジ検出せずにstart信号をカウンタのリセットにじかに接続してしまうと，8サイクルの間リセットし続けます．

制御系では，カウンタの値から演算系の制御信号を生成します．

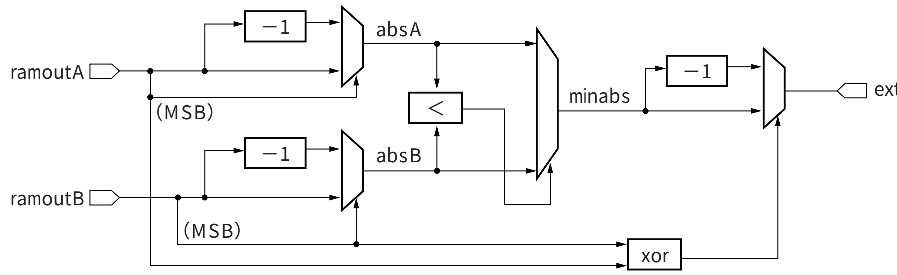


図4
EXT回路のブロック図
絶対値をとって小さいほうを選択し，符号を調整する．

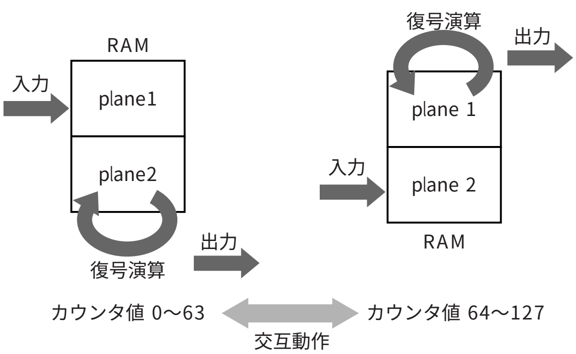


図5　受信信号の取り込みと復号の並列処理
RAMをアドレスによってplane1/2に分けて作業する．カウンタが0〜63の間はplane1に受信信号を取り込み，plane2で復号する．64〜127の間はplane2に受信信号を取り込み，plane1で復号する．

演算系では，アーキテクチャのところで説明した復号演算を行います．EXT回路のブロック図を**図4**に示します．絶対値を取って小さいほうを選択し，符号を調整，と計算式どおりの実装です．加算器とEXT回路の入出力はオーバフローを防ぐため，受信信号に対してMSB側に1ビット拡張しています．

● メモリを2プレーンに分割

すべての回路は，7ビット(0〜127)のカウンタにより動作を制御しています．

DCMでクロックを8てい倍しているので，clk8の8サイクルごとに1個のシリアル・データを受信します．したがって，clk8の64サイクルごとに1セット(8個)のデータを受信することになります．

8個のデータがそろわないと，復号演算を実行することはできません．データがそろうまでの間は，直前の8個のデータについて復号を行います．具体的には，RAMをアドレスによってplane1かplane2に分けて作業します．カウンタが0〜63の間はplane1に受信信号を取り込み，plane2で復号します．64〜127の間はplane2に受信信号を取り込み，plane1で復号します．これを交互に行うことによって受信信号の取り込みと復号を並列的に処理します(**図5**)．

● プログラミングの要領でタイミング設計

この回路の動作シーケンスを考える作業は，低級言語のプログラミングとほぼ同じです．演算タイム・テーブルを**表2**に示します．各サイクルにおけるRAMのアクセス，演算内容，制御信号の値などを示します．

8の倍数のタイミングで定期的に受信信号を取り込んでいます．それと同時に，もう一方のポートから作業変数(w_0，w_1，w_2，w_3，w_4)の初期化も行います．

復号演算は受信信号の取り込みタイミングを避けて行います．演算のシーケンスを考えるに当たって，RAMのWrite after Read動作によって生ずる次の制約に注意します．

書き込みアドレス ＝ 次のサイクルの
　　　　　　　　　　一方の読み出しアドレス

ここで，次のサイクルに読み出されるデータは，直前に書き込んだデータではなく，書き込みを行う以前のデータです．例えば，**表2**の最初の演算は，

$w_0 = w_3 + y_0$　　　(No. 2)
$w_1 = w_0 + y_1$　　　(No. 3)

ですが，No.3で読み出されるw_0は，No.2の演算結果$w_3 + y_0$ではなく，書き込みを行う以前の初期値'0'です．直前の演算結果を次の演算に使用したい場合は，1サイクル以上待つ必要があります．例えば，**表2**のNo.5〜No.7では，演算結果$w_2 + y_3$を参照したいため，1サイクル待っています．

$w_3 = w_2 + y_3$　　　(No. 5)
(idle)　　　(No. 6)
$w_4 = \mathrm{ext}(w_3, r_1)$　　　(No. 7)

表2　演算タイム・テーブル

変数名
アドレス

W：ライト
R ：リード
C ：クリア（'0'を書き込む）

No.	RAM アクセス																										式	制　御					備　考
	plane1													plane2																			
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	10	11	12	13	14	15	16	17	18	19	1A	1B	1C							
	y0	y1	y2	y3	r0	r1	c0	c1	w0	w1	w2	w3	w4	y0	y1	y2	y3	r0	r1	c0	c1	w0	w1	w2	w3	w4		weA	adrA	weB	adrB	select	
0														W								C						1	10	1	18	ZERO	データ格納および作業領域のクリア
1																												0	00	0	0B	ZERO	―
2	R								W			R															$w_0=w_3+y_0$	0	01	1	08	ADD	加　算
3		R							R	W																	$w_1=w_0+y_1$	0	02	1	09	ADD	
4			R							R	W																$w_2=w_1+y_2$	0	03	1	0A	ADD	
5				R							R	W															$w_3=w_2+y_3$	0	05	1	0B	ADD	
6																												0	05	0	0B	ZERO	―
7						R						R	W														$w_4=\mathrm{ext}(w_3, r_1)$	0	00	1	0C	EXT	ext
8															W								C					1	11	1	19	ZERO	データ格納および作業領域のクリア
9																												0	04	0	08	ZERO	―
10					R				R	W																	$w_1=\mathrm{ext}(w_0, r_0)$	0	04	1	09	EXT	ext
11					R					R	W																$w_2=\mathrm{ext}(w_1, r_0)$	0	05	1	0A	EXT	
12						R					R	W															$w_3=\mathrm{ext}(w_2, r_1)$	0	03	1	0B	EXT	
13																												0	03	0	0B	ZERO	―
14				R								R	W														$w_4=w_3+y_3$	0	02	1	0C	ADD	加　算
15			R						W				R														$w_0=w_4+y_2$	0	00	1	08	ADD	
16																W								C				1	12	1	1A	ZERO	データ格納および作業領域のクリア
17																												0	01	0	09	ZERO	―
18		R								R	W																$w_2=w_1+y_1$	0	00	1	0A	ADD	加　算
19	R										R	W															$w_3=w_2+y_0$	0	06	1	0B	ADD	
20																												0	06	0	0B	ZERO	―
21							R					R	W														$w_4=\mathrm{ext}(w_3, c_0)$	0	07	1	0C	EXT	ext
22								R	W				R														$w_0=\mathrm{ext}(w_4, c_1)$	0	06	1	08	EXT	
23							R		R	W																	$w_1=\mathrm{ext}(w_0, c_0)$	0	00	1	09	EXT	
24																	W								C			1	13	1	1B	ZERO	データ格納および作業領域のクリア
25																												0	07	0	0A	ZERO	―
26								R			R	W															$w_3=\mathrm{ext}(w_2, c_1)$	0	03	1	0B	EXT	ext
27																												0	03	0	0B	ZERO	―
28				R								R	W														$w_4=w_3+y_3$	0	02	1	0C	ADD	加　算
29			R						W				R														$w_0=w_4+y_2$	0	01	1	08	ADD	
30		R							R	W																	$w_1=w_0+y_1$	0	00	1	09	ADD	
31	R									R		W															$w_3=w_1+y_0$	0	04	1	0B	ADD	
32																		W								C		1	14	1	1C	ZERO	データ格納および作業領域のクリア
33																												0	04	0	0B	ZERO	―
34					R							R	W														$w_4=\mathrm{ext}(w_3, r_0)$	0	05	1	0C	EXT	ext
35						R			W				R														$w_0=\mathrm{ext}(w_4, r_1)$	0	05	1	08	EXT	
36						R			R	W																	$w_1=\mathrm{ext}(w_0, r_1)$	0	04	1	09	EXT	
37					R					R		W															$w_3=\mathrm{ext}(w_1, r_0)$	0	00	1	0B	EXT	
38																												0	00	0	0B	ZERO	―
39	R										W	R															$w_2=w_3+y_0$	0	00	1	0A	ADD	加　算
40																			W							C		1	15	1	1C	ZERO	データ格納および作業領域のクリア
41																												0	01	0	0C	ZERO	―
42		R							W				R														$w_0=w_4+y_1$	0	02	1	08	ADD	加　算
43			R						R	W																	$w_1=w_0+y_2$	0	03	1	09	ADD	
44				R						R		W															$w_3=w_1+y_3$	0	07	1	0B	ADD	
45																												0	06	0	18	ZERO	―
46					W		R															R					$r_0=c_0$	0	00	1	04	ADD	コピー
47																												0	00	0	00	ZERO	―
48																				W						C		1	16	1	1C	ZERO	データ格納および作業領域のクリア
49																												0	07	0	18	ZERO	―
50						W		R														R					$r_1=c_1$	0	00	1	05	ADD	コピー
51																												0	09	0	06	ZERO	―
52						W	R			R																	$r_1=\mathrm{ext}(w_1, c_0)$	0	0B	1	05	EXT	ext
53					W	R						R															$r_0=\mathrm{ext}(w_3, r_1)$	0	0A	1	04	EXT	
54					R			W			R																$c_1=\mathrm{ext}(w_2, r_0)$	0	08	1	07	EXT	
55							W	R	R																		$c_0=\mathrm{ext}(w_0, c_1)$	0	00	1	06	EXT	
56																					W					C		1	17	1	1C	ZERO	データ格納および作業領域のクリア
57																												0	05	0	0A	ZERO	―
58						R					R																w_2+r_1	0	04	0	08	ZERO	事後値
59					R				R																		w_0+r_0	0	07	0	09	ZERO	
60								R		R																	w_1+c_1	0	06	0	0B	ZERO	
61							R					R															w_3+c_0	0	00	0	00	ZERO	
62																												0	00	0	00	ZERO	―
63																												0	00	0	00	ZERO	

最近のCPUは，並列実行(out of order)のためにデータの依存関係が小さくなるように命令を並べます．それがこのハードウェアではまったく逆になるのは，おもしろい相違です．

演算の最後に加算器から事後値が出るので，MSBをレジスタに取り込みます．出力信号が4ビットいっせいに変わるようにリタイミングします．

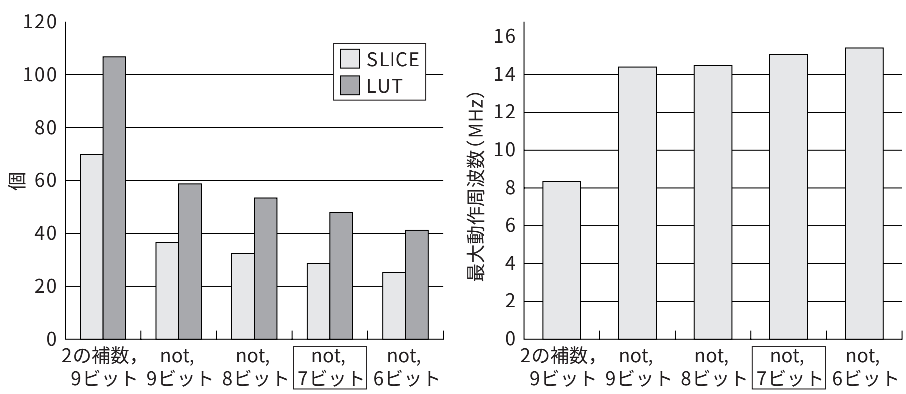
● Cプログラムと RTL シミュレーションが合わない！

回路をHDLで記述し，論理シミュレーションを実行しました．演算器の入出力値がC言語シミュレーションの値と同じように推移するかどうかの確認です．数回の修正を経た後，計算経過が一致するようになりました．後は配布データ 20,000個を復号して，エラー・レートがCと一致すれば大成功です．

ところが，エラー・レートがわずかに異なることに気が

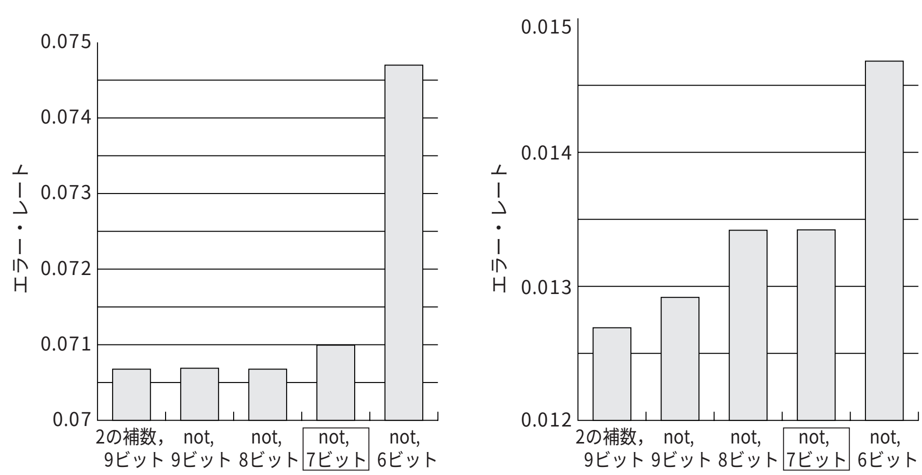
表3
合成結果とエラー・レート

					採用回路		
回路仕様	符号反転	2の補数	not	not	not	not	基準アルゴリズム
	演算ビット幅	9	9	8	7	6	
合成結果	SLICE［個］	70	36	33	29	25	
	LUT［個］	107	59	53	47	41	
	最大動作周波数［MHz］	8.286	14.49	14.67	15.09	15.37	
エラー・レート	S/N 0dB	0.0707	0.0707	0.0707	0.0710	0.0747	0.0701
	S/N 3dB	0.0127	0.0129	0.0134	0.0134	0.0147	0.0125
	S/N 6dB	0.0002	0.0003	0.0003	0.0002	0.0008	0.0002
	S/N 9dB	0	0	0	0	0	0
	S/N 100dB	0	0	0	0	0	0



(a) SLICE数　(b) 動作周波数

図6
合成結果の比較
not回路を使用した場合のほうがSLICE数が約半分，最大動作周波数が1.75倍で有利．



(a) S/N＝0dB　(b) S/N＝3dB

図7
エラー・レートの比較
7ビットまではほとんど違いが見られない．

つきました．RAMのアクセスが複雑なので，ここを重点的にチェックしてみましたがわかりません．大量の時間を消費したあげく，C言語の記述にミスがあったことがわかりました．事後値の正負を判別する不等号で0をどちらに含めるか，という問題です．MSBで正負を判別するならば，0は正として扱うべきでしたが逆になっていました．事後値が0になる頻度はまれで，かつHDL記述にまちがいがあるという思い込みから，バグの発見に時間がかかりました．上流工程のバグを下流になってから修正するのはたいへんだという教えを再認識しました．

● 論理合成と回路のチューンアップ

回路レベルでチューンアップ可能なところとして，次の2ヵ所が考えられます．

- EXT回路
- 演算ビット幅

EXT回路は，3個の符号反転回路を含んでいます．昨年のコンテスト受賞作品で使われていた手法(4)，すなわち2の補数回路の代わりにnot回路を用いる方法により回路規模の減少が期待できます．演算ビット幅を減らすことも回路規模の削減に有効です．ただし，どちらの方法も計算精度を犠牲にします．これらの影響を調べるため，回路仕様を変えて論理合成とシミュレーションを行いました．

論理合成ツールとしてXilinx社のISE WebPACK 7.1.04iのXSTを使い，ターゲット・デバイスはXC3S50-5-VQ100に設定しました．結果を**表3**と**図6**，**図7**に示します．表中の最大動作周波数は外部クロックの値です．内部回路は8てい倍されたクロックで動作します．**図7**の縦軸（エラー・レート）の範囲は0を含んでいないことに注意します．

符号反転回路に2の補数回路とnot回路を用いた場合を比較します．合成結果を比較すると，not回路を使用した場合のほうがSLICE数が約半分，最大動作周波数が1.75倍で有利です．エラー・レートを比較するとほとんど違いが見られません．よってnot回路を採用することにしました．

次に演算ビット幅について検討します．前述のとおりMSB側に1ビット拡張したので，下位ビットを削らない場合のビット幅は9ビットです．エラー・レートを比較すると，6ビットまで減らすと急激に復号性能が悪くなることがわかります．よって演算ビット幅を7ビットに決めました．このとき，エラー・レートは基準アルゴリズムと比べてS/Nが0dBで1%，3dBで7%悪化しています．

実際にはアプリケーションに応じて復号性能の可否を評価するべきですが，今回の場合は適当な指標がないため，主観で判断しました．比較基準用の50入力XORと採用した復号回路の合成結果の比較を**表4**に示します．SLICE数は50入力XORの3倍未満です．

ここまで作業が完了したとき，一服感と同時に，若干不安な気持ちに襲われました．最初からわかっていたことですが，動作周波数が低くて見劣りするためです．しかしせっかくのコンテストなのだから特徴のある（クセの強い）回路のほうがおもしろいと思い，この回路に決めました．

● シミュレーション

チューンアップ後の回路のシミュレーション結果を**図8**と**図9**に示します．受信信号$\{y_0, y_1, y_2, y_3, r_0, r_1, c_0, c_1\} = \{-35, -36, 14, -50, 0, -35, -111, 14\}$を復号したときのシミュレーション波形です．LSB側を2ビット削除するので，回路内部では$\{y_0, y_1, y_2, y_3, r_0, r_1, c_0, c_1\} = \{-9, -9, 3, -13, 0, -9, -28, 3\}$として扱われます．**図8**の上側の実線の枠内がRAMの出力＝演算器の入力，破線の枠内が演算器の出力，下側の実線の枠内がRAMの入力です．**図10**に具体的な数値計算の一部を示します．**図8**のシミュレーション結果と一致することが確認できます．

事後値のMSBの取得とリタイミングのようすを**図9**に示します．

● FPGAに実装して実機動作させる

手持ちの設備が貧弱なので，FPGAの実機実験はなるべく簡素にすることにしました．FPGAボードは，コンテスト参加者向けに貸し出されていた「Spartan-II E300評価キット」と本誌2004年1月号の付属ボードを利用しました．以降，前者をSpartan-IIEボード，後者をSpartan-3ボードと呼びます．

表4　50入力XOR回路との比較

	50入力XOR	デコーダ回路
SLICE［個］	10	29
LUT［個］	17	47
UNIT_AREA	(1)	2.9
遅　延［ns］	10.669	66.29
UNIT_DELAY	(3)	18.6

注：FPGAの専用機能ブロックの使用数は含んでいない．

2

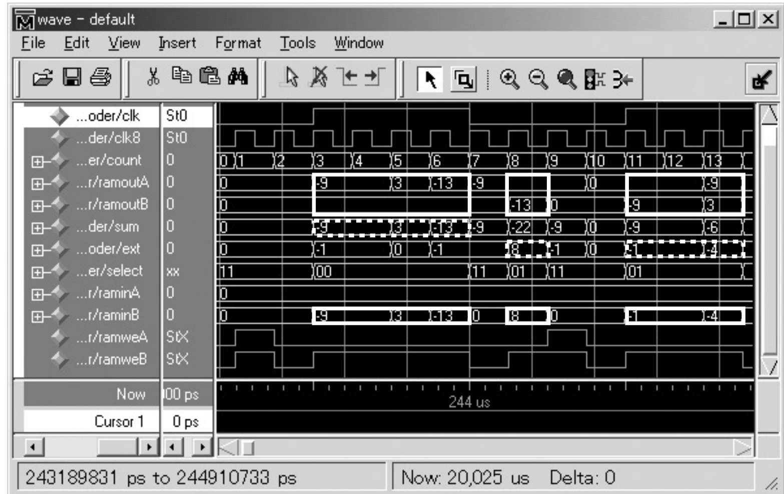


図8
シミュレーション結果
上側の白い実線の枠内がRAMの出力＝演算器の入力，白い破線の枠内が演算器の出力，下側の白い実線の枠内がRAMの入力．

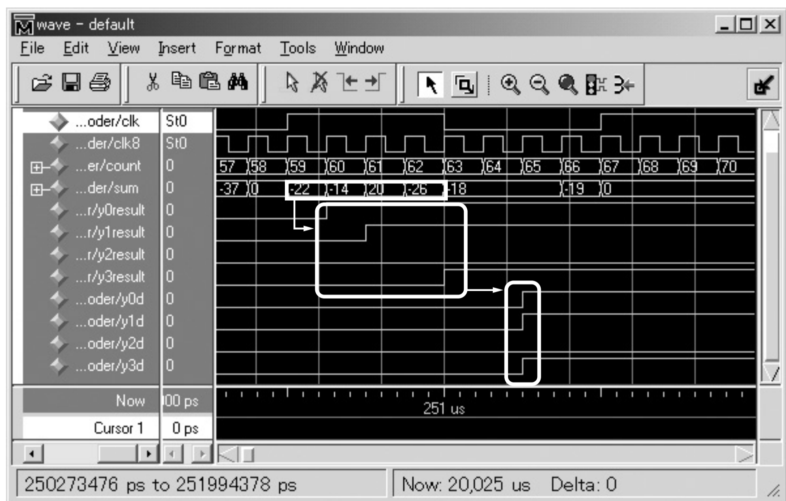


図9
事後値のMSBの取得とリタイミングのようす
sumの符号を取得する．出力が4ビット同時に変化するようにタイミングをそろえる．

$$
\begin{aligned}
w_0 &= w_3 + y_0 & -9 &= 0 + (-9)\\
w_1 &= w_0 + y_1 & -9 &= 0 + (-9)\\
w_2 &= w_1 + y_2 & 3 &= 0 + 3\\
w_3 &= w_2 + y_3 & -13 &= 0 + (-13)\\
\\
w_4 &= \mathrm{ext}\ (w_3,\ r_1) & 8 &= \mathrm{EXT}(-13,\ -9)\\
w_1 &= \mathrm{ext}\ (w_0,\ r_0) & -1 &= \mathrm{EXT}(-9\ ,\ \ 0)\\
w_2 &= \mathrm{ext}\ (w_1,\ r_0) & -1 &= \mathrm{EXT}(-9\ ,\ \ 0)\\
w_3 &= \mathrm{ext}\ (w_2,\ r_1) & -4 &= \mathrm{EXT}(\ 3\ ,\ -9)\\
&\vdots & &\vdots
\end{aligned}
$$

図10　数値計算の一部

Spartan-IIEボードにエラー評価用回路を書き込み，Spartan-3ボードに復号回路を書き込みました．評価システムのブロック図を**図11**に示します．図中の灰色のブロックがSpartan-3ボード，それ以外がSpartan-IIEボードに属します．

動作の概要を説明します．クロックはSpartan-IIEボード上の66MHzの信号を16分周して作ります．問ROMから復号回路にstart信号とrxin信号を与えます．解ROMと復号回路の出力を比較し，エラーを評価します．解ROMのvalid信号が"H"の間エラーをカウントし，エラー数をSpartan-IIEボード上の7セグメントLEDに表示します．

ROMの容量のつごう上，配布データ20,000個分の評価を1度に実行することができません．このためデータを{8,000個，8,000個，4,000個}の3個に分割して評価しました．その結果，エラー数はRTLシミュレーションと一致し，復号回路が機能していることが確認できました．評価のようすを**写真1**に示します．上側のボードがSpartan-IIEボード，下側のボードがSpartan-3ボードです．中央の基板は上下のボードを電気的に接続するためのもので，

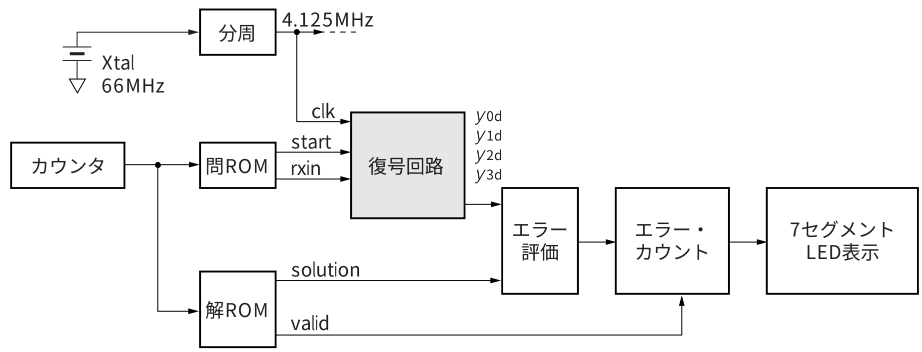


図11
評価システムのブロック図
Spartan-IIEボードをエラー評価用回路，Spartan-3ボードを復号回路に割り当てる．図中の灰色のブロックがSpartan-3ボード，それ以外がSpartan-IIEボード．

電源のパスコン以外に電子部品は載っていません．作成した部品はこの基板だけです．

● エピローグ

FPGAでDCMなどのクロック生成技術を利用したことがなかったため，設計段階から実機できちんと動作するだろうかと心配していました．しかし，トラブルもなく実機での実験は丸一日で完了しました．便利な技術が使えるようになって，今後の電子工作にも役に立ちそうです．

今回，課題を通して事後確率や最尤（さいゆう）推定の一端について知りました．Webサイトで調べて知ったことですが，音声認識にも利用されているようです．（技術と比べて）抽象度の高い数学だからこそ応用範囲も広いのだと思います．数学的な基礎を固めておけば，自分の専門外の技術に対してももっと見通しが良くなるだろう，と勉強する気になってきました．願わくば三日坊主になりませんように…．

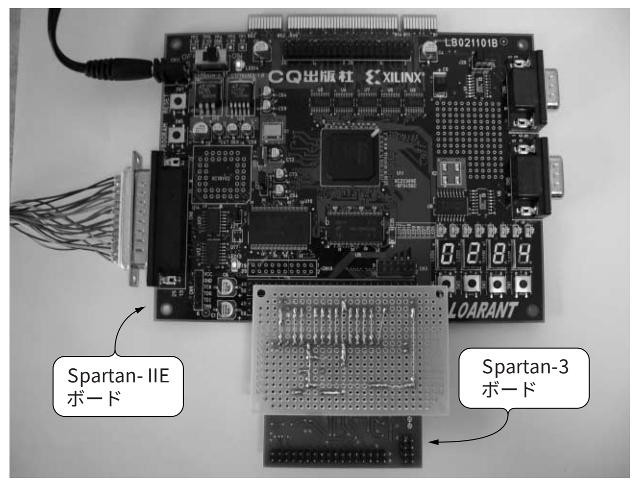


写真1　評価システム
上側のボードがSpartan-IIEボード，下側のボード（裏向き）がSpartan-3ボード．中央の基板は上下のボードを電気的に接続するためのもの．

2

参考・引用*文献

(1) 和田知久；2次元積符号用繰り返し型デコーダ設計仕様書，Design Wave Magazine，2005年11月号．
(2) Spartan-3 FPGAでのブロックRAMの使用，アプリケーションノート XAPP463，Xilinx社．
(3) Spartan-3 FPGAにおけるディジタルクロックマネージャ（DCM）の使用，アプリケーションノート XAPP462，Xilinx社．
(4) 山本健児，横田雅也；ディジタルFMレシーバ設計のトライ&エラー，Design Wave Magazine，2005年6月号．

たなか・ひろまさ

<筆者プロフィール>
半導体メーカ勤務．入社3年め．大学院の専攻はシステム制御・信号処理．最近はGPUを使った汎用計算GPGPUに興味を持っている．